

4 JEJU INN

포딩 매뉴얼
SSAFY 6 반 3 조

2023.02

I.개요

1. 프로젝트 개요

'JEJUINN'은 제주도 게스트 하우스 스텝 매칭 서비스입니다. 코로나 이후로 여행 수요가 증가하며 해외여행 대신 제주도 게스트 하우스에서 스텝을 하며 '제주도 한 달 살기'를 하는 사람 또한 많아졌습니다.

하지만, 스텝을 구인, 구직하는 과정은 일반적인 채용 과정과 달라 '특정 플랫폼 카페'에서 직접 글을 올리고 문자를 보내는 방식으로 채용이 진행됩니다. 이러한 환경 속에서 발생할 수 있는 불편함, 정보의 부족을 해결하기 위해 'JEJUINN'은 스텝을 하고 싶은 사람에게는 게스트 하우스와 제주도 여행지에 대한 다양한 정보를 게스트 하우스 사장님에게는 손쉬운 구인 글 작성, 스텝 관리, 화상 면접, AI 를 통한 스텝 추천 기능을 제공합니다.

'JEJUINN'을 통해 사용자가 스텝 구인 및 구직을 더 간단하고 명료하게 하며, 게스트 하우스 스텝이 추천하는 제주도의 여러 관광지 정보를 얻을 수 있기를 기대합니다.

2. 프로젝트 사용 도구

- 이슈 관리
 - Jira
- 형상 관리
 - Gitlab
- 커뮤니케이션
 - Notion
 - Mattermost
- 디자인
 - Figma
- UCC
 - 프리미어
 - 모바비
- CI/CD
 - Jenkins
 - Docker

3. 개발 환경

- VS Code: 1.75.1
- IntelliJ: 223.8214.52

- **JVM: 11.0.1**
- **Node.js: 18.13.0**
- **Server: AWS EC2 Ubuntu 20.04 LTS**
- **DB: MySQL 8.0.31**


4. 외부 서비스

- **Amazon Web Service**
 - **Relational Database Service(RDS)**
'Application.yml'에 해당 내용 있음, **과금 발생주의**
 - **Amazon Simple Storage Service(Amazon S3)**
'Application.yml'에 해당 내용 있음, **과금 발생주의**
- **OAUTH**
 - **KAKAO API**
'env.local'에 해당 내용 있음
 - **NAVER API**
'env.local'에 해당 내용 있음
 - **GOOGLE API**
'env.local'에 해당 내용 있음

II. 빌드

5. 환경변수

- frontend/.env.local



```
// 카카오 프론트엔드 키
REACT_APP_KAKAO_REST_API_KEY
REACT_APP_KAKAO_REDIRECT_URI
REACT_APP_KAKAO_JAVASCRIPT_KEY

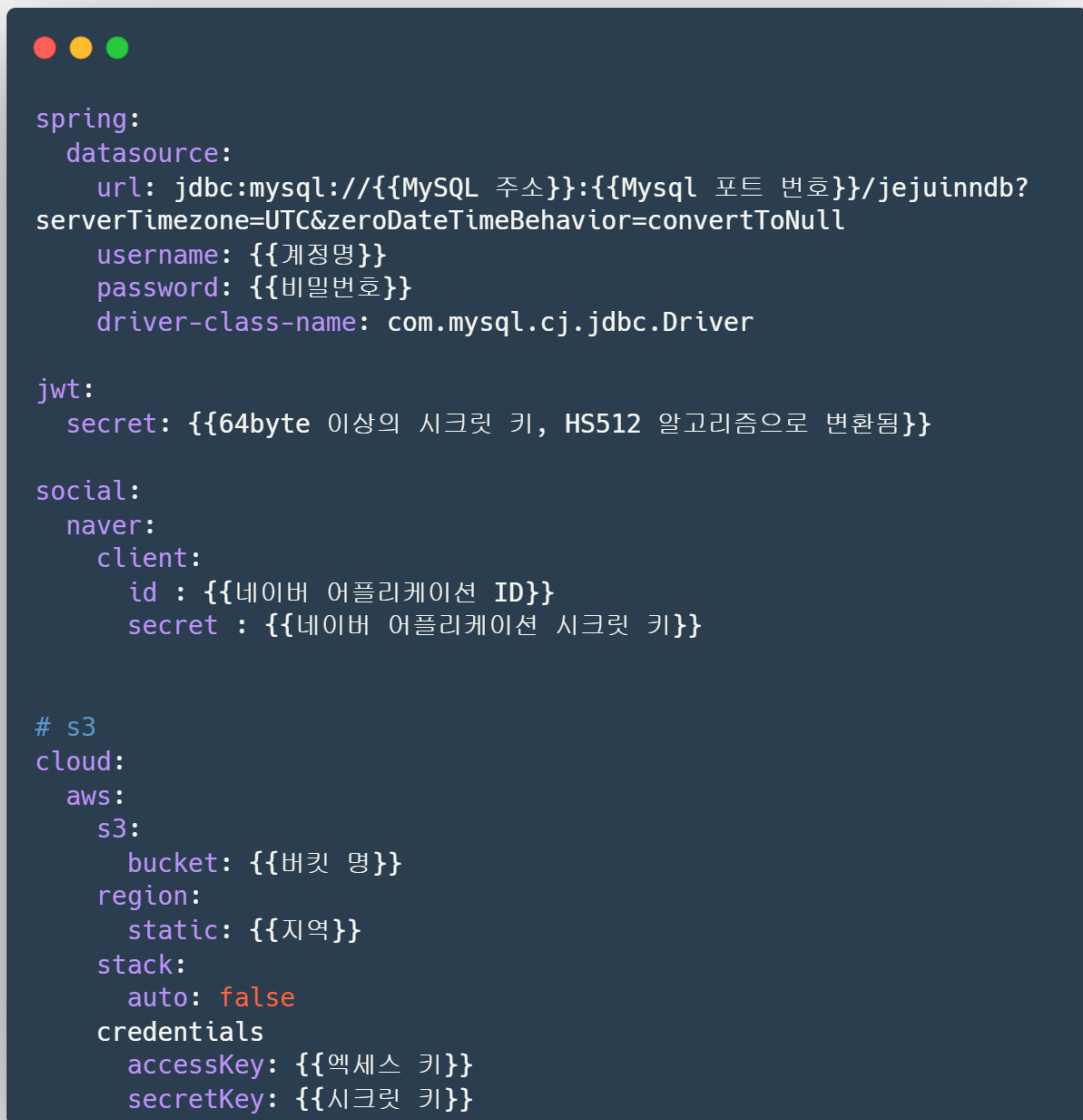
// 구글 프론트엔드 키
REACT_APP_GOOGLE_REDIRECT_URI
REACT_APP_GOOGLE_CLIENT_ID
REACT_APP_GOOGLE_CLIENT_SECRET

// 네이버 프론트엔드 키
REACT_APP_NAVER_CLIENT_ID
REACT_APP_NAVER_SECRET
REACT_APP_NAVER_CALLBACK_URL
REACT_APP_NAVER_AUTH_CALLBACK_URL

// 네이버 지도 프론트엔드 키
REACT_APP_NAVER_MAP_ID
REACT_APP_NAVER_MAP_SECRET
```

[그림 1] .env.local

- backend/src/main/resources/application.yml

A screenshot of a code editor showing the content of the application.yml file. The code is written in a light blue font on a dark blue background. It defines Spring datasource properties, JWT secret, social login properties for Naver, and AWS S3 credentials. The code includes placeholders like {{MySQL 주소}}, {{Mysql 포트 번호}}, and {{계정명}}.

```
spring:
  datasource:
    url: jdbc:mysql://{{MySQL 주소}}:{{Mysql 포트 번호}}/jejuinndb?
serverTimezone=UTC&zeroDateTimeBehavior=convertToNull
    username: {{계정명}}
    password: {{비밀번호}}
    driver-class-name: com.mysql.cj.jdbc.Driver

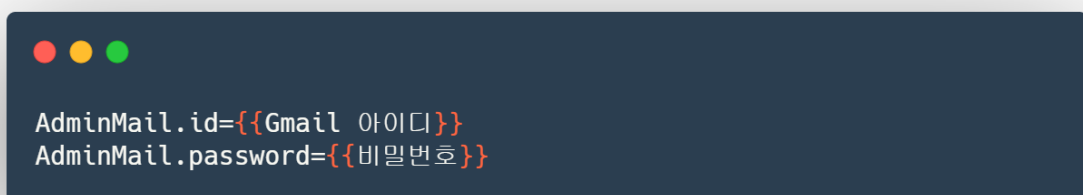
jwt:
  secret: {{64byte 이상의 시크릿 키, HS512 알고리즘으로 변환됨}}

social:
  naver:
    client:
      id : {{네이버 어플리케이션 ID}}
      secret : {{네이버 어플리케이션 시크릿 키}}

# s3
cloud:
  aws:
    s3:
      bucket: {{버킷 명}}
      region:
        static: {{지역}}
      stack:
        auto: false
      credentials
        accessKey: {{엑세스 키}}
        secretKey: {{시크릿 키}}
```

[그림 2] application.yml

- backend/src/main/resources/email.properties

A screenshot of a code editor showing the content of the email.properties file. The code is written in a light blue font on a dark blue background. It defines AdminMail.id and AdminMail.password with placeholders like {{Gmail 아이디}} and {{비밀번호}}.

```
AdminMail.id={{Gmail 아이디}}
AdminMail.password={{비밀번호}}
```

[그림 3] email.properties

6. 배포

- Nginx 설정

```
server {
    listen 443 ssl default_server;
    listen [::]:443 ssl default_server;

    root /var/www/html;

    index index.html index.htm index.nginx-debian.html;

    server_name www.jejuinn.com jejuinn.com;

    ssl_certificate /etc/letsencrypt/live/jejuinn.com/fullchain.pem;
    ssl_certificate_key
/etc/letsencrypt/live/jejuinn.com/privkey.pem;

    location / {
        proxy_pass http://127.0.0.1:3000;
    }

    location /sim {
        proxy_pass http://127.0.0.1:5000;
    }

    location ~ ^/(swagger|webjars|configuration|swagger-
resources|v2|csrf|api|auth) {
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $host;
        proxy_pass http://127.0.0.1:8080;
    }
}
server {
    if ($host = www.jejuinn.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    if ($host = jejuinn.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

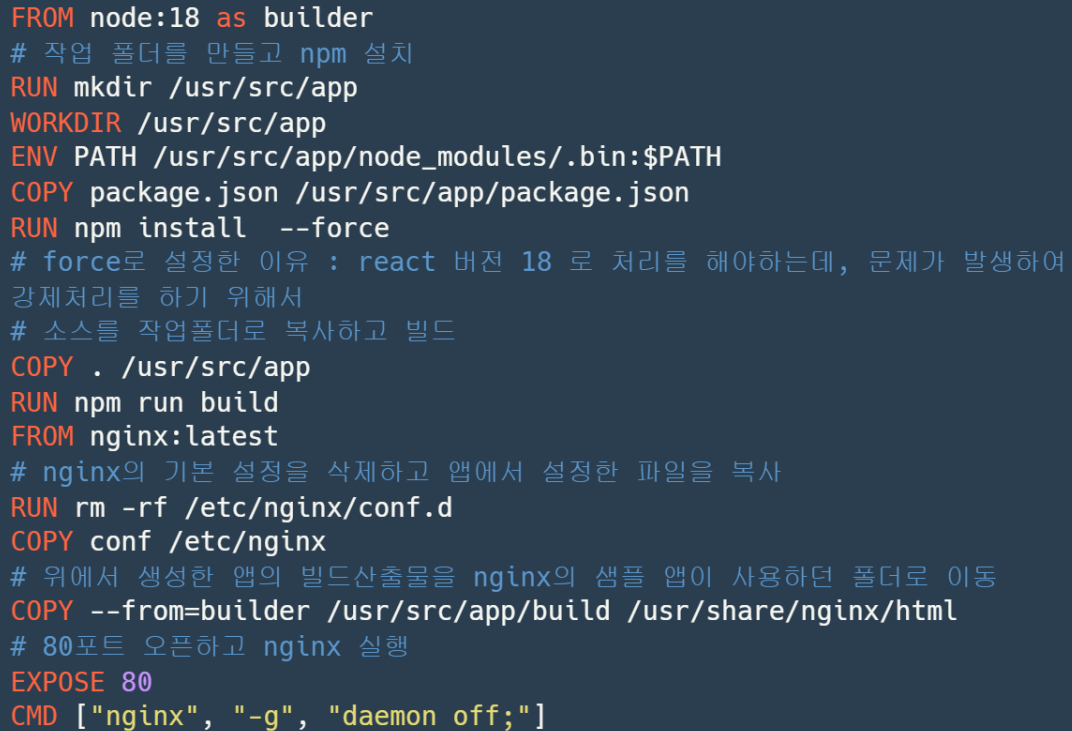
    listen 80;

    server_name jejuinn.com www.jejuinn.com;
    return 404; # managed by Certbot
}
```

[그림 4] Nginx 설정 파일

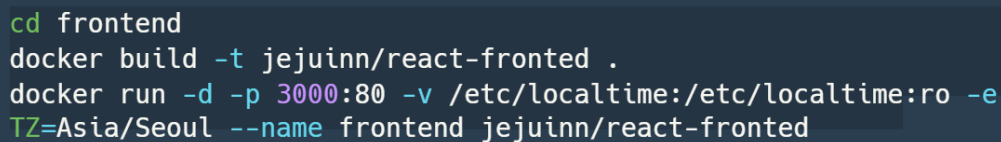
- Docker 빌드

- React



```
FROM node:18 as builder
# 작업 폴더를 만들고 npm 설치
RUN mkdir /usr/src/app
WORKDIR /usr/src/app
ENV PATH /usr/src/app/node_modules/.bin:$PATH
COPY package.json /usr/src/app/package.json
RUN npm install --force
# force로 설정한 이유 : react 버전 18 로 처리를 해야하는데, 문제가 발생하여
# 강제처리를 하기 위해서
# 소스를 작업폴더로 복사하고 빌드
COPY . /usr/src/app
RUN npm run build
FROM nginx:latest
# nginx의 기본 설정을 삭제하고 앱에서 설정한 파일을 복사
RUN rm -rf /etc/nginx/conf.d
COPY conf /etc/nginx
# 위에서 생성한 앱의 빌드산출물을 nginx의 샘플 앱이 사용하던 폴더로 이동
COPY --from=builder /usr/src/app/build /usr/share/nginx/html
# 80포트 오픈하고 nginx 실행
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

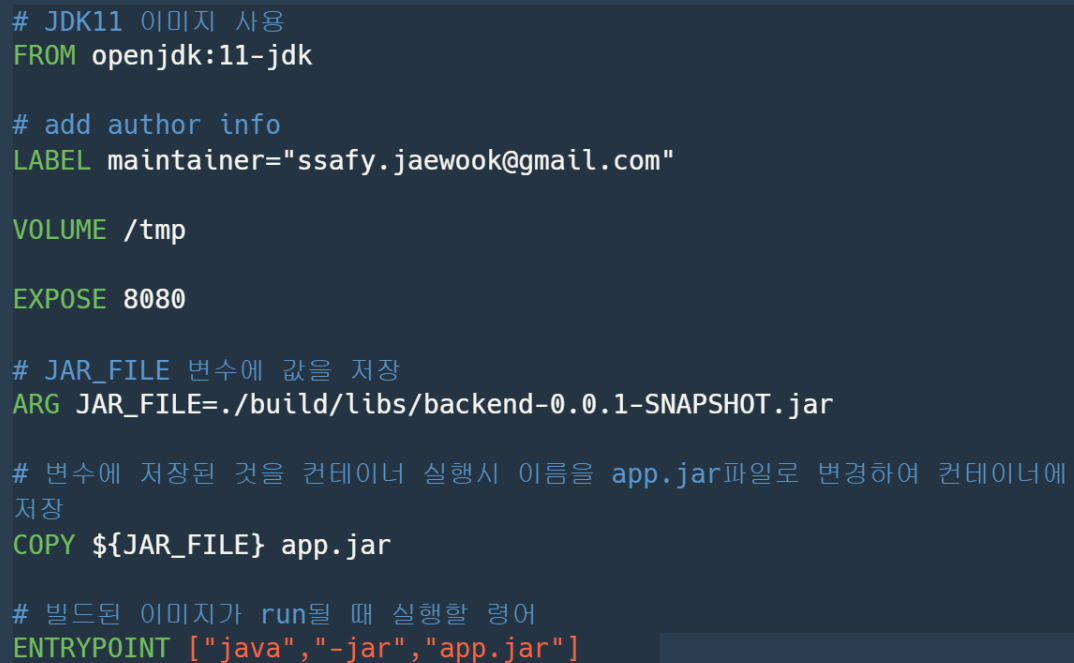
[그림 5] 리액트 Dockerfile



```
cd frontend
docker build -t jejuinn/react-fronted .
docker run -d -p 3000:80 -v /etc/localtime:/etc/localtime:ro -e
TZ=Asia/Seoul --name frontend jejuinn/react-fronted
```

[그림 6] 리액트 도커 이미지 빌드 및 실행

- Spring boot



```
# JDK11 이미지 사용
FROM openjdk:11-jdk

# add author info
LABEL maintainer="ssafy.jaewook@gmail.com"

VOLUME /tmp

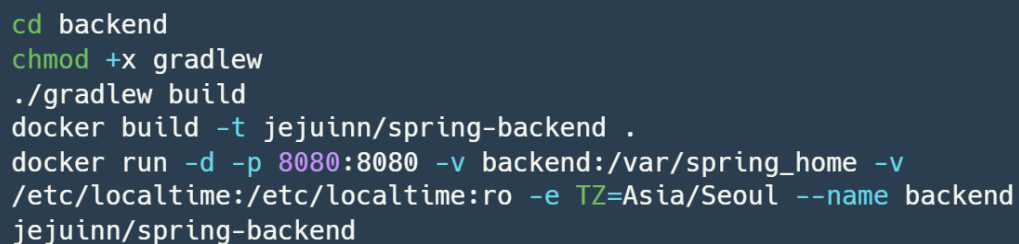
EXPOSE 8080

# JAR_FILE 변수에 값을 저장
ARG JAR_FILE=./build/libs/backend-0.0.1-SNAPSHOT.jar

# 변수에 저장된 것을 컨테이너 실행시 이름을 app.jar파일로 변경하여 컨테이너에 저장
COPY ${JAR_FILE} app.jar

# 빌드된 이미지가 run될 때 실행할 령어
ENTRYPOINT ["java", "-jar", "app.jar"]
```

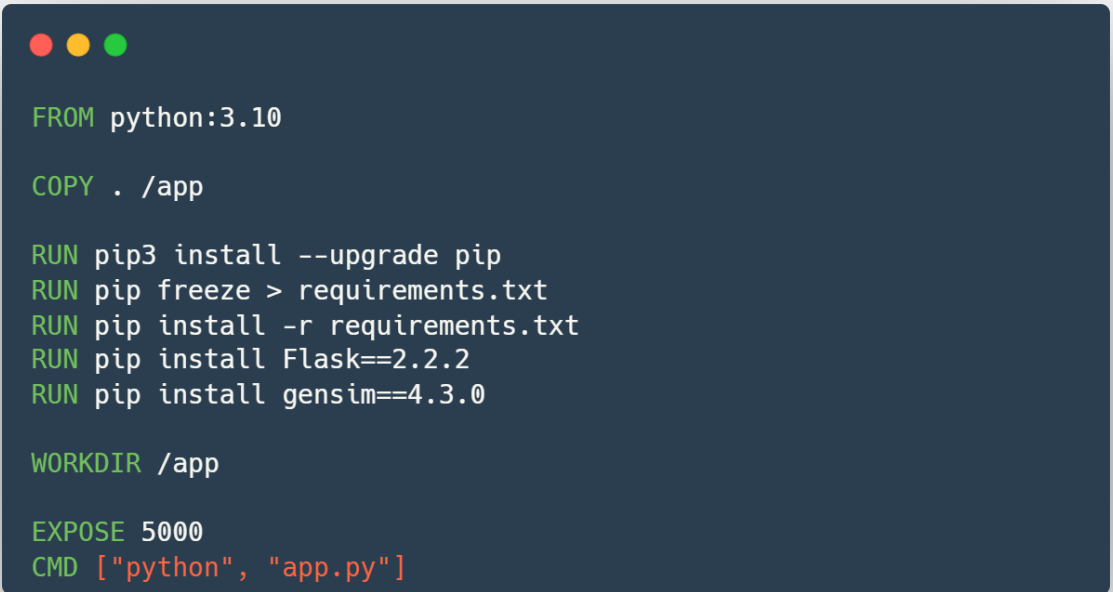
[그림 7] 스프링 부트 Dockerfile



```
cd backend
chmod +x gradlew
./gradlew build
docker build -t jejuinn/spring-backend .
docker run -d -p 8080:8080 -v backend:/var/spring_home -v /etc/localtime:/etc/localtime:ro -e TZ=Asia/Seoul --name backend jejuinn/spring-backend
```

[그림 8] 스프링 부트 도커 이미지 빌드 및 실행

- Flask



```
FROM python:3.10

COPY . /app

RUN pip3 install --upgrade pip
RUN pip freeze > requirements.txt
RUN pip install -r requirements.txt
RUN pip install Flask==2.2.2
RUN pip install gensim==4.3.0

WORKDIR /app

EXPOSE 5000
CMD ["python", "app.py"]
```

[그림 9] 플라스크 Dockerfile



```
cd backend-flask
docker build -t jejuinn/flask-backend .
docker run -d -p 0.0.0.0:5000:5000/tcp --name backend-flask
jejuinn/flask-backend
```

[그림 10] 플라스크 도커 이미지 빌드 및 실행

각 프로젝트의 Dockerfile 은 프로젝트 루트 디렉토리에 있습니다.

7. 외부 서비스

A. 카카오 API

- i. 내 애플리케이션 -> 애플리케이션 추가하기로 이동합니다.
- ii. 앱 이름과 사업자명을 입력합니다.
- iii. 콘솔 -> 카카오 로그인으로 이동합니다.
- iv. 상태를 ON 으로 변경합니다.
- v. Redirect URI 를 추가합니다.
- vi. 동의 항목을 설정합니다.

B. 네이버 API

- i. 애플리케이션 등록 (API 이용신청)으로 이동합니다.
- ii. 애플리케이션 이름을 작성합니다.
- iii. 사용 API 에서 검색을 추가합니다.
- iv. 로그인 오픈 API 서비스 환경을 PC 웹으로 설정합니다.
- v. 로그인 오픈 API 서비스 환경에서 서비스 URL 과 네이버 로그인 Callback URL 을 등록합니다.

C. 구글 API

- i. 프로젝트를 생성으로 이동합니다.
- ii. 프로젝트 이름과 조직 및 위치를 입력합니다.
- iii. 구글 클라우드 플랫폼 -> 생성한 프로젝트 -> API 및 서비스 -> OAuth 동의 화면으로 이동합니다.
- iv. 웹 주소, 웹 이름, 관리자 이메일을 입력합니다.

D. AWS S3

- i. S3 -> 버킷 만들기로 이동합니다.
- ii. 버킷 이름, AWS 리전을 입력합니다.
- iii. S3 의 퍼블릭 액세스 차단을 해제합니다.
- iv. 버킷 버전 관리는 비활성화, 기본 암호화 또한 비활성화로 설정합니다.
- v. 버킷을 생성합니다.
- vi. 버킷 정책을 생성합니다.
 - 1. Select Type of Policy: S3 Buckey Policy
 - 2. Effect: Allow
 - 3. Principal: *
 - 4. Actions: GetObject (조회만 가능하도록 getObject 권한만 체크했습니다.)
 - 5. Amazon Resource Name (ARN): arn:aws:s3:::{버킷이름}/*

- vii. 생성한 정책의 json 값을 복사합니다.
- viii. 이전에 생성한 버킷 -> 버킷 정책 -> 편집으로 이동합니다.
- ix. 이전 단계에서 복사한 정책의 json 값을 붙여넣고 변경 사항 저장을 합니다.

E. AWS RDS

- i. RDS -> 데이터베이스 생성으로 이동합니다.
- ii. 데이터 베이스 생성 방식은 표준 생성으로 선택합니다.
- iii. 엔진 옵션에서 엔진 유형은 MySQL, 버전은 8.0.28 으로 선택합니다.
- iv. 템플릿을 프리티어로 선택합니다.
- v. DB 인스턴스 식별자, 마스터 사용자 이름, 마스터 암호를 입력합니다.
- vi. 연결에서 퍼블릭 액세스를 예로 변경합니다.
- vii. 추가구성에서 초기 데이터베이스 이름(본 프로젝트에서는 'jejuinndb')을 입력합니다.
- viii. 데이터베이스를 생성합니다.
- ix. 생성한 데이터베이스에서 VPC 보안 그룹을 선택합니다.
- x. 보안 그룹 -> 인바운드 규칙 -> 인바운드 규칙 편집으로 이동합니다.
- xi. 인바운드 규칙
 - 1. 유형 : MYSQL/Aurora
 - 2. 프로토콜 : TCP
 - 3. 포트범위 : 3306
- xii. 규칙을 저장합니다.